TABLE DES MATIERES

Chapitre I

Systèmes de numération, Arithmétique binaire & Codes

A. Systèmes de numération

A.1. Généralités

Nos calculs sont toujours effectués dans le système de numération décimal. Un micro-ordinateur ne peut pas utiliser ce système, les circuits qui le constituent ne permettent pas de distinguer 10 états. Les calculs effectués par la machine sont donc tributaires de la nature particulière des circuits. ceux-ci ne pouvant prendre que deux états représentés par 0 et 1, le système de numération utilisé ne comportera que deux symboles 0 et 1, c'est le système de numération binaire. Pour étudier les systèmes de numération utilisés dans la technique du calcul digital, nous allons nous référer à un système qui nous est familier : le système décimal.

A.2. Le système décimal

La base du système est 10: Elle représente le nombre de symboles dont on dispose pour représenter les nombres dans ce système. Les 10 symboles du système décimal sont les chiffres successifs allant de 0 à 9. Ces chiffrent représentent une quantité par eux-même mais peuvent en représenter une autre par la position qu'ils occupent dans un nombre. La place occupée par le chiffre dans un nombre représente son rang.

Exemple :

$$(9817{,}45)_{10} = 9.10^3 + 8.10^2 + 1.10^1 + 7.10^0 + 4.10^{-1} + 5.10^{-2}$$
$$= 9000 + 800 + 10 + 7 + 0{,}4 + 0{,}05$$

Cas général :

$$A = a_n a_{n-1}.....a_0, a_{-1}.....a_{-n}$$
$$= a_n.10^n + a_{n-1}.10^{n-1} + ...+ a_0.10^0 + a_{-1}.10^{-1} + ...a_{-n}.10^{-n}$$
$$= \left(\sum_{i=-n}^{n} a_i.10^i \right)_{10}$$

$a_i \subset \{0,\ 1,.....9\}$

Pour une base B quelconque : $A = \left(\sum_{i=-n}^{n} a_i.B^i \right)_B$

$a_i \subset \{0,\ 1,.....B-1\}$

A.3. Le système binaire

La base de ce système est 2 (B =2). Cela signifie que l'on dispose que 2 symboles pour représenter des nombres: Ces symboles sont 0 et 1. Un nombre binaire sera donc constitué d'une suite de 0 et de 1.

On a donné à ces symboles le nom de bits, contraction de binary-digit .

Exemple :

$$(1011,011)_2 = 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3}$$
$$= 8 + 0 + 2 + 1 + 0 + 0,25 + 0,125$$
$$= (11,375)_{10}$$

A.4. Conversion décimal-binaire

L'équivalent binaire d'un nombre décimal s'obtient en divisant successivement le nombre décimal par 2 jusqu'au moment ou le quotient est nul. Ce dernier quotient et les restes des divisions successives, regroupés de droite à gauche, forment le nombre binaire cherché.

Exemple : Soit à convertir $(21)_{10}$

	21 | 2	
r_0 →	1 | 10 | 2	
r_1 →	0 | 5 | 2	
r_2 →	1 | 2 | 2	
r_3 →	0 | 1 | 2	
r_4 →	1 | 0	quotient nul

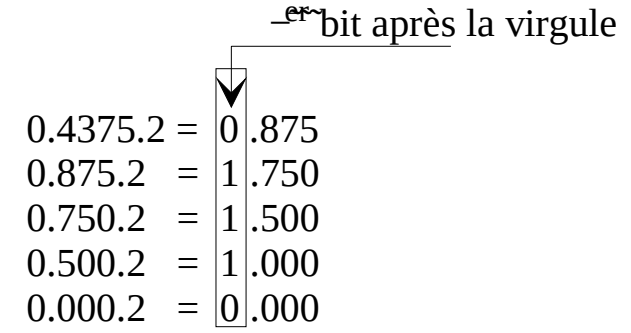
D'où $(21)_{10} = (10101)_2$

Vérification:

$$(21)_{10} = 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0$$
$$= 16 + 0 + 4 + 0 + 1$$
$$= (21)_{10}$$

Dans le cas de nombres fractionnaires on multiplie la partie fractionnaires par 2: Si après la multiplication un "1" apparaît à gauche de la virgule, on ajoute "1" à la fraction binaire en formation, si après la multiplication, c'est un "0" qui apparaît, on ajoute un "0".

Exemple : Soit à convertir $(0.4375)_{10}$



D'où $(0.4375)_{10} = (0,01110)_2$

Vérification :

$$(0.4375)_{10} = 0.2^{-1} + 1.2^{-2} + 1.2^{-3} + 1.2^{-4}$$
$$= 0 + 0.25 + 0.125 + 0.0625$$
$$= (0.4375)_{10}$$

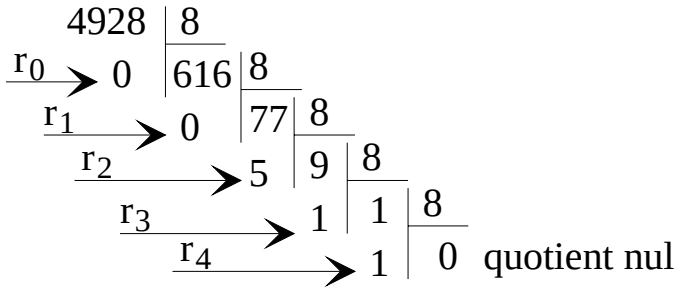
A.5. Le système octal : (B = 8 et $a_i \subset \{0, 1, ...7\}$)

Exemple :

$$(476)_8 = 4.8^2 + 7.8^1 + 6.8^0$$
$$= 256 + 56 + 6$$
$$= (318)_{10}$$

A.6. Conversion décimal-octal

Exemple : Soit à convertir $(4928)_{10}$



D'où $(4928)_{10} = (11500)_8$

A.7. Conversion octal-binaire

La base du système octal étant égale à la puissance troisième de 2, l'équivalent binaire d'un nombre octal, s'obtient facilement en écrivant la suite des équivalents binaires de ses chiffres exprimés chacun au moyen de trois bits. ($B = 2^3 = 8$)

Exemple : $(4173)_8$

Chiffres octaux	**4**	**1**	**7**	**3**
Equivalent binaire	**100**	**001**	**111**	**011**

Ainsi $(4173)_8 = (100001111011)_2$

De même pour le passage inverse

Exemple :

$$(\underline{100}\ \underline{110}\ \underline{000}\ \underline{001})_2$$
$$\downarrow\quad\downarrow\quad\downarrow\quad\downarrow$$
$$(4\quad 6\quad 0\quad 1)_8$$

A.8. Le système hexadécimal

On dispose de 16 symboles pour représenter un nombre hexadécimal. Ces symboles sont les dix chiffres du système décimal auxquels on a ajouté 6 signes supplémentaires. Par conversion il a été admis d'utiliser les six premières lettres de l'alphabet. Les 16 symboles du système sont alors : 0, 1, ..., 9, A, B, C, D, E et F.

Exemple :

$$(4CA2)_{16} = 4.16^3 + C.16^2 + A.16^1 + 2.16^0$$
$$= 16384 + 3072 + 160 + 2$$
$$= (19618)_{10}$$

A.9. Conversion décimal-hexadécimal

Exemple : Soit à convertir $(92628)_{10}$

92628	16	
r_0 → 4	5789	16
r_1 = D → 13	361	16
r_2 → 9	22	16
r_3 → 6	1	16
r_4 → 1	0	quotient nul

D'où $(92628)_{10} = (169D4)_{16}$

Application : Convertissez en hexadécimal ces nombres décimaux :

a. 75 b. 314 c. 2048 d. 25619 e. 4095

A.10. Conversion hexadécimal-binaire

La base du système hexadécimal étant égale à la puissance quatrième de 2, l'équivalent binaire d'un nombre hexadécimal s'obtient en écrivant la suite des équivalents binaires de chacun de ses signes exprimés au moyen de 4 bits. ($B = 2^4 = 16$)

Exemple : $(4CA2)_{16}$

Chiffres hexadécimaux	**4**	**C**	**A**	**2**
Equivalent binaire	**0100**	**1100**	**1010**	**0010**

D'où $(4CA2)_{16} = (0100110010100010)_2$

De même pour l'inverse

Exemple :

$$(\underset{\downarrow}{1001}\ \underset{\downarrow}{1000}\ \underset{\downarrow}{0001})_2$$
$$(\ 9 \quad 8 \quad 1\ )_{16}$$

Application : Convertissez en hexadécimal les nombres binaires :

a. 10110 b. 10001101 c. 100100001001 d. 1111010111 e. 1011111

A.11. Virgule fixe et virgule flottante

Dans un ordinateur les entiers sont représentés en virgule fixe ou fixée. La virgule binaire est fixée dans la mesure qu'elle se trouve à la fin de chaque mot binaire et ainsi chaque valeur représentée est un entier positif ou négatif.
Exemple de codification d'entiers relatifs sur un mot mémoire UNIVAC 1100 (le mot comprend 36 bits avec un bit de signe dans la position 35)

Les entiers positifs exemple : $(12)_{10} = (1100)_2$

0.	....	**0**	**1**	**1**	**0**	**0**

35 0

Les entiers négatifs exemple : $(-12)_{10}$. On utilise le complément à "1"

1.	....	**1**	**0**	**0**	**1**	**1**

35 0

L'entier 0

0.	....	**0**

35 0

Ou

1.	....	**0**
35		0

En calcul scientifique, il est souvent nécessaire de calculer avec des nombres très grands ou très petits. Ainsi on représente le nombre par une mantisse et un exposant; et on parle de représentation en virgule flottante.

Exemple : $(4900000)_{10} = (0.49.10^{7})_{10}$

$$x = ((0, A)_{10} .10^{(E)})_{10}$$

De même

$$(0.00023)_{10} = (0.23.10^{-3})_{10}$$

Exemple de codification des nombres rationnels sur UNIVAC 1100

Soit $x = (0.1275)_{10} = (0.001100)_2 = (0.11)_2.2^{126-128} = (0.75)_{10}.2^{-2}$

0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	....	**0**
35	**34**	**Exposant**						**27**	**26**	**Mantisse**			**0**

signe

Soit $x = (12)_{10} = (1100)_2 = (0.11)_2.2^{132-128} = (0.75)_{10}.2^{4}$

0	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	....	**0**
35	**34**	**Exposant**						**27**	**26**	**Mantisse**			**0**

signe

Remarque

Les nombres rationnels binaires sont enregistrés en normalisés (le premier chiffre significatif de la mantisse est un "1" logique).

La codification des nombres rationnels négatifs se fait par complémentation à "1".

Application

Dans la plus part des micro-ordinateurs, les adresses des emplacements mémoires sont exprimés en hexadécimal. Ces adresses sont des nombres séquentiels qui identifient chacune des cases mémoires.

a. Un certain micro- ordinateur peut stoker des nombres de 8 bits dans chacune de ses cases mémoires. Si l'intervalle des adresses mémoires va de 0000_{16} à $FFFF_{16}$, dites combien cet ordinateur a de cases de mémoires.

b. Un autre micro-ordinateur possède 4096 emplacements en mémoire. Donnez l'intervalle de ses adresses exprimées en hexadécimal.

B. Arithmétique binaire

Quand un calculateur effectue une opération arithmétique, deux contraintes interviennent:

- les circuits travaillent sur des nombres qui ont toujours la même longueur (FORMAT); par exemple, dans une machine de 8 bits (ou 8 éléments binaires ou 8 e.b), le nombre 10111 doit être complété par zéros 00010111;
- les circuits ne manipulent que deux nombres à la fois; pour calculer S = X +Y +Z, ils effectuent X +Y = s puis s + Z = S.

En effectuant une addition, une retenue (**CARRY**) peut apparaître; dans une soustraction, il peut y avoir une retenue (**BORROW**); enfin si en manipulant des nombres trop grands pour le format, on obtient un dépassement de capacité (**OVERFLOW**)différent d'une retenue; il doit être signalé par un drapeau (**FLAG**).

B.1. Les quatre opérations

Les mêmes règles de calcul s'appliquent dans tous les systèmes de numération; l'arithmétique binaire à celle des nombres décimaux.

Addition		**Soustraction**	
	retenue		***retenue***
0 + 0 = 0	0	0 - 0 = 0	0
0 + 1 = 1	0	0 - 1 = 1	1
1 + 1 = 0	1	1 - 1 = 0	0
1 + 0 = 1	0	1 - 0 = 1	0

Multiplication	**Division**		
		quotient	***reste***
0 . 0 = 0	0 : 0	impossible	
0 . 1 = 0	0 : 1	0	1
1 . 1 = 1	1 : 1	1	0
1 . 0 = 0	1 : 0	impossible	

Exemple

Addition						**Soustraction**					
Décimal	***Binaire***					***Décimal***	***Binaire***				
	r	+1	+1	+1			***r***	-1	-1		
11		1	0	1	1	19	1	0	0	1	1
+ 07	+	0	1	1	1	- 05	-		1	0	1
= 18	= 1	0	0	1	0	= 14	= 0	1	1	1	0

Multiplication	
Décimal	***Binaire***
11 x 5 = 55	1011 x 101 1011 1011. = 110111

Des circuits électroniques assez simples réalisent des additions. On ramène une soustraction à une addition à l'aide d'une représentation adéquate des nombres négatifs.

Pour faire une multiplication, il faut effectuer les produits partiels, des décalages et des additions. Quant à la division, elle n'est pas faite comme en numération décimale. Elle se ramène à une suite de multiplication et de comparaison.

A retenir

Les quatre opérations arithmétiques nécessitent trois opérateurs (additionneur, multiplicateur, comparateur) et un circuit de décalage.

B.2. Les nombres négatifs

Afin d'utiliser un additionneur comme soustracteur, il faut une représentation convenable des nombres négatifs.

a. Représentation avec valeur absolue et le signe (module & signe)

On adopte la convention 0 (+), 1 (-) pour le signe:

+35 = **0** 100011 - 35 = **1** 100011

il faut, pour le signe un traitement spécial, différent de celui des e.b de la valeur absolue, ce qui ne permet pas une utilisation commode de l'additionneur en soustracteur.

b. Représentation par le complément restreint (complément à 1)

On obtient le complément restreint CR(X) d'un nombre X en remplaçant ses 0 par des 1 et ses 1 par des 0. Si n est le nombre d'e.b de X, on a $X + CR(X) = 2^n - 1$ ainsi que le montre l'exemple suivant (n = 4):

X = 1101
CR(X) = 0010

X + CR(X) = 1111 = $2^4 - 1$

On déduit que : $-X = CR(X) + 1 - 2^n$, ce qui montre que CR (X) + 1 représente le nombre négatif - X à condition de ne pas tenir compte du chiffre de poids 2^n dans l'expression numérique de CR (X) + 1, c'est-à-dire de conserver les n e.b de poids $\leq 2^{n-1}$

La quantité CV (X) = CR (X) + 1 s'appelle le complément vrai. En écrivant :

+ 13	= 000	1101
CR(13) + 1	= 111	0011
13 + CV(13)	=.....000	0000

on a bien éliminé le terme de poids 24 et la relation précédente donne :

$$13 + CV(13) = 0000$$

ou -13 = CV(13). A condition de manipuler des nombres de longueur fixe.

***c. Représentation par le complément vrai** (Complément à 2)*

C'est la représentation utilisée dans tous les calculateur. Nous venons de voir comment on aboutit naturellement à cette représentation des nombres négatifs.

$$CV(X) = CR(X) + 1 \leftrightarrow -X$$

+ 0	0	0000	- 0	0	0000
+ 1	0	0001	- 1	1	1111
+ 2	0	0010	-2	1	1110
.....	....	.....	.....	.....	.....
+ 15	0	1111	-15	1	0001
			-16	1	0000

e.b.de signe

Quelques remarques doivent être faites :

- on une seule représentation du zéro, ce qui n'est pas le cas avec le complément restreint;
- de la relation $X + CR(X) = 2^n - 1$ on déduit $CV(X) = 2^n - X$ et CV[CV(X)] = X;
- la représentation de +16 (0 10000) ne convient pas car elle a 6 e.b. au lieu de 5; en outre, on obtient CV(16) = 101111+1 = 110000 qui a 6 e.b. et n'est pas contenue avec la représentation des nombres -1, ..., -15;
- Par contre la représentation (10000) de -16 convient car elle a 5 éléments binaires (e.b.), elle contenue avec les représentation des nombres -1,.....,-15 et parce qu'elle redonne la valeur absolue correcte:

$$CV[CV(16)] = CV(10000) = 01111 + 1 = 10000 = 16$$

B.3. Problèmes liés à la longueur des nombres

Nous venons de montrer que les circuits traitant des nombres de n e.b. (n = 8 par exemple) peuvent manipuler tous les nombres entre -2^{n-1} et $2^{n-1}-1$ (-128 et +127) à condition que les résultats partiels ne sortent pas de cet intervalle.

a. Addition de deux nombres positifs

En ajoutant 2 nombres > 0 (e.b. de signe, 0) on peut obtenir un résultat dont l'e.b. de signe est 1 (bien que le résultat soit positif) si le résultat est hors de l'intervalle autorisée (+127, -128 dans l'exemple suivant) :

```
+ 49   0   0 1 1 0 0 0 1          + 49   0   0 1 1 0 0 0 1
+ 33   0   0 1 0 0 0 0 1          +88    0   1 0 1 1 0 0 0
--------  --------------------     --------  ----------------------
+ 82   0   1 0 1 0 0 1 0          + 137  1   0 0 0 1 0 0 1
       +      82  ⟶                          ↑   ⟶ CV(119) = - 119
                                  dépassement de capacité ( 8 ème e.b. à 1)
```

Les calculateurs utilisent un indicateur de dépassement de capacité (OVERFLOW FLAG) $f_d = 1$ si dans le résultat R l'e.b. de signe S_R est à 1 alors que dans les deux nombres positifs A et B les éléments binaires de signes S_A et S_B sont à 0. On écrit $f_d = \bar{S}_A.\bar{S}_B.S_R$

b. Addition de deux nombres négatifs

En additionnant deux nombres < 0 représentés par leurs compléments vrais (e.b. de signe, 1) on peut obtenir un résultat dont l'e.b. de signe (rang n) est 0 (bien que le résultat soit négatif). En outre il y a toujours une retenue car l'e.b. de rang (n+1) est toujours à 1.

```
 - 32      1   1 1 0 0 0 0 0          - 32      1   1 1 0 0 0 0 0
 - 31      1   1 1 0 0 0 0 1          -127      1   0 0 0 0 0 0 1
---------    --------------------     ---------  ----------------------
 - 63   1  1   1 0 0 0 0 0 1          -159  1   0   1 1 0 0 0 0 1
retenue ⟵      CV(63) ⟵              retenue ⟵      ⟶ + 97 → FAUX
            -      63                          dépassement car 8ème e.b. à 0
```

Le 9ème e.b. est toujours à 1 de même que les 10ème, 11ème....etc. Dans la représentation par le complément vrai; par exemple :

```
+ 11 : .....0 0 0 0 1 0 1 1
CV(X) : .....1 1 1 1 0 1 0 1
```

Ce 9ème e.b. est donc une retenue. Des deux exemples précédents, le premier donne un résultat correct, car il est dans l'intervalle (+127, -128); le second donne un résultat faux à cause du dépassement de capacité (-159 est hors de l'intervalle, +127, -128). L'indicateur de dépassement $f_d = 1$ car, dans le résultat, l'e.b. de signe S_R (8ème e.b.) est à 0 alors que dans les 2 nombres négatifs les e.b. de signe S_A et S_B (8ème e.b.) sont à 1; on écrit $f_d = S_A.S_B.\overline{S_R}$ avec la notation présentée à la fin du paragraphe précédent. On réunit ces 2 résultats en écrivant $f_d = \overline{S_A}.\overline{S_B}.S_R$ ou $S_A.S_B.\overline{S_R}$

c. Addition d'un nombre positif et d'un nombre négatif

Cela revient à effectuer la soustraction A - B. Pour cela, on forme :

$$A + CV(B) = A + 2^n - B = 2^n + A - B$$

- Si $A \geq B$ on a $A - B \geq 0$ et $A + CV(B) \geq 2^n$; donc :

$$A - B = A + CV(B) - 2^n$$

La soustraction se ramène à une addition dans laquelle on néglige la retenue qui apparaît ($n+1^{ième}$ e.b. de poids 2^n) : la présence de ce terme dans une comparaison de conclure à $A \geq B$.

- Si $A < B$ on a $A - B < 0$ et $A + CV(B) < 2^n$

$$A + CV(B) = 2^n - (B - A) = CV(B - A).$$

Il n y a pas de retenue ($n+1^{ième}$ e.b. de poids 2^n), ce qui, dans une comparaison, permet de conclure à A < B et dans une soustraction de prendre le complément vrai du résultat obtenu qui est CV(B-A); en effet, on a :

$$CV(CV(B-A)) = 2^n - CV(B-A) = 2^n - [2^n - (B-A)] = B-A$$

9		0 1 0 0 1	7		0 0 1 1 1
-5		1 1 0 1 1	-10		1 0 1 1 0
------		-----------------	--------		------------------
+4	1	0 0 1 0 0	- 3		1 1 1 0 1
rejeter		+ 4			CV (3)
					- 3

C. LES CODES

C.1. Généralités

Le codage permet la spécification des caractères (par exemple caractère numérique ou alphanumérique) en utilisant d'autres symboles. Les codes ont été utilisés pour des raisons de sécurité; ce qui a permis la protection des messages secrets. Dans les ordinateurs la codification permet de spécifier les caractères en utilisant simplement les symboles binaires 0 et 1. Le choix parmi plusieurs codes binaires dépend de la fonction à accomplir.

Il y a des codes qui se prêtent bien pour des opérations arithmétiques. D'autres sont meilleurs parce qu'ils sont plus efficace en donnant plus d'information avec le minimum de bit. L'importance qu'apporte également les codes réside dans la détection et la correction d'erreur; en effet les codes donnent les possibilités à un calculateur de déterminer si un caractère qui a été codé et transmis a été reçu correctement; et s'il existe une erreur de la corriger. Le codage en lui même est un sujet très vaste, on présentera dans ce qui suit les codes les plus familiers.

C.2. Code binaire naturel, code octal et hexadécimal

Le système de numération binaire est considéré comme un code binaire naturel, il permet en effet de représenter des nombres sous forme binaire; le tableau 1 présente le code binaire naturel pour des nombres compris entre 0 et 16.

Tableau 1

décimal	C.B.Naturel
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

Un nombre binaire à 16 bits (1110100101101110 par exemple), utilisé par un ordinateur, est difficilement lu et reconnu. En plus la conversion en décimal n'est

pas facile à obtenir. Afin de permettre une expression facile des nombres binaires on utilise le codage octal ou hexadécimal. La majorité des calculateurs utilisent en effet le codage hexadécimal pour représenter les nombres binaires.

Exemple: Soit à exprimer le nombre binaire suivant en octal et en hexadécimal:

10111101010001011101

Ce nombre est représenté en octal comme suit:

010	111	101	010	001	011	101
2	7	5	2	1	3	5

ou en hexadécimal:

1011	1101	0100	0101	1100
B	D	4	5	C

C.3. Décimal codé binaire BCD « Binary Coded Decimal »

a. BCD-(8421)

C'est un code utilisé pour exprimer en binaire un nombre décimal compris entre 0 et 9 (voir tableau 2)

Tableau 2

décimal	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ce code nécessite 4 bits pour représenter un caractère décimal, Il est moins efficace que le système décimal, mais il a l'avantage d'être sous forme binaire pour être facilement exploitable sur ordinateur.

Quelques exemples de ce code:

Décimal	BCD (8421)
0110 0010	62
0101 0011	53
0110 1001 0001	691
0011 0111 0101 1001	3759

Dans les opérations arithmétiques on rencontre des difficultés dans l'utilisation des codes BCD.

Exemple:

Décimal	Binaire	BCD(8421)	
8	1000	1000	
+ 7	+ 0111	0111	
15	1111	1111	Ce caractère n'est pas accepté dans le BCD (15: 0001 0101)

Un additionneur spécial est nécessaire pour assurer les opérations dans le code BCD.

b. Autres codes BCD

En plus du code BCD (8421) qui est le plus utilisé, il existe plusieurs autres codes utilisant des bits pour représenter les nombres décimaux. Ces codes peuvent être à 4 bits, 5 bits, 7 bits (biquinaire) et même à 10 bits. On présentera dans ce qui suit quelques codes à 4 et à 5 bits qui sont les plus utilisés.

b.l. BCD excédent 3 (exc3)

Il représente également les nombres décimaux de 0 à 9. Le tableau 3 illustre le code BCD exc3 qui est obtenu en ajoutant 3 (0011) à chaque valeur du code BCD (8421). Contrairement au code BCD (8421) l'excédent 3 est un code qui n'est pas pondéré.

Tableau 3

décimal	BCD(8421)	Excèdent 3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Exemple: représentation de quelques nombres en excédent 3

Décimal	Excédent 3
5	1000
30	0110 0011
24	0101 0111
361	0110 1001 0100
8496	1011 0111 1100 1001

Le code excédent 3 permet d'assurer les opérations arithmétique d'une façon plus facile (en comparaison avec le code BCD (8421)).

Exemple1 :

```
   4  0111
 + 2  0101
 ------------
   6  1100     (pas de retenue
 -      11      on retranche donc 3)
 ------------
      1001
```

Exemple 2:

```
   3              0110
 + 9            + 1100
 ----           ------
  12            1 0010     (Retenue = l
                + 0011           on ajoute 3)
                ------
                 0101
```

Il existe donc quelques règles à suivre pour accomplir l'opération d'addition (exemple: ajouter 3 à chaque digit s'il existe une retenue), Ces étapes lorsqu'ils sont programmées peuvent être assurées automatiquement sur le calculateur, ce qui rend l'excédent 3 plus apprécié pour les opérations arithmétiques.

b.2. Codes BCD à 4 bits

Ces codes sont très nombreux, quelques exemples de codes pondérés sont représentés dans le tableau 4.

Tableau 4

Décimal	4221	5421	7421	74-2-1	84-2-l
0	0000	000	000	0000	000
1	0001	001	001	0111	111
2	0010	010	010	0110	110
3	0011	011	011	0101	101
4	0110	100	100	0100	100
5	0111	000	101	1010	011
6	010	001	110	001	010
7	011	010	111	000	001
8	110	011	001	111	000
9	111	100	010	110	111

Exemple 1: Soit à convertir en décimal le nombre suivant (représenté dans le code BCD (5421)): 0011 1000 1010

0011 = 0 x 5 + 0 x 4 + 1 x 2 + 1 x 1 = 3
1000 = 1 x 5 + 0 x 4 + 0 x 2 + 0 x 1 = 5
1010 = 1 x 5 + 0 x 4 + 1 x 2 + 0 x 1 = 7

En se référant au tableau 4 on peut vérifier:

$$(0011\ 1000\ 1010)_{5421} = (357)_{10}$$

Exemple 2: Soit à convertir en décimal le nombre suivant (représenté dans le code BCD(74-2-1)): 0101 0110 1111

0101 = 0 x 7 + 1 x 4 - 0 x 2 - 1 x 1 = 3
0110 = 0 x 7 + 1 x 4 - 1 x 2 - 0 x 1 = 2
1111 = 1 x 7 + 1 x 4 - 1 x 2 - 1 x 1 = 8

En se référant à au tableau 4 on peut vérifier:

$$(0011\ 1000\ 1010)_{74\text{-}2\text{-}1} = (328)_{10}$$

b.3. Codes BCD à 5 bits

Dans le tableau 5 on présente deux codes BCD à 5 bits; le premier code, appelé 2 parmi 5, est un code pondéré qui est utilisé pour la détection d'erreur dans les systèmes de communication. Le deuxième code (51111) est un code pondéré; il a l'avantage d'être auto-complémenté.

Tableau 5

Décimal	2 parmi 5	51111
0	00011	0000
1	00101	00001
2	00110	00011
3	01001	00111
4	01010	01111
5	01100	10000
6	10001	11000
7	10010	11100
8	10100	11110
9	11000	11111

Exemple: Soit à exprimer le nombre 285 dans les codes:

a) 2 parmi 5

$(285)_{10}$ = 00110 10100 01100

b) BCD (51111)

$(285)_{10}$ = 00011 11110 10000

C.4. Les codes alphanumérique

En plus des codes binaires spécifiant les nombres décimaux de 0 à 9, il existe plusieurs codes qui peuvent représenter à la fois les caractères numériques et alphabétiques. Ces codes sont appelés alphanumériques. parmi ces codes le ASCII (American Standard Code for Information Interchange) qui est le plus utilisé.

ASCII est un code standard à 7 bits (généralement le 8ème bit pour la parité). Ce code permet la spécification des nombres décimaux, des caractères alphabétiques et quelques caractères spéciaux. Dans le tableau 6 figure tous les caractères du code ASCII et leur valeur correspondante en binaire et en hexadécimale.

Exemple : Un message transmit ("START 1." par exemple) se présente au niveau de l'ordinateur en code ASCII de la manière suivante:

S	T	A	R	T	1	.
1010011	1010100	1000001	1010010	1010100	0110001	0101110

Plus simplement on peut représenter en hexadécimal

S	T	A	R	T	1	.
53	54	41	52	54	31	2E

Tableau 6

C.5. Codes de détection et de correction d'erreurs

Dans divers cas de transmission de données, les informations binaires peuvent être erronées. Ceci peut être dû à un mauvais fonctionnement d'un circuit électronique ou encore à la longue distance de transmission. La détection et la correction d'erreurs est un domaine d'étude très vaste dans les transmissions de données.

a. Bit de parité

La détection d'erreurs la plus populaire est celle utilisant un bit de parité.

Exemple:

bit de parité (impaire)	bit de parité (paire)
010001 1	010101 1
donnée	donnée

Lorsqu'un mot est reçu, sa parité est testé (paire ou impaire: choix fait à l'émission) et il est accepté comme correct s'il satisfait le test. Malheureusement ce simple test n'est pas suffisant pour localiser le bit erroné; en plus deux erreurs dans un même mot passent inaperçues. Le tableau 7 présente le code BCD (8421) avec les bits de parité pour les deux cas (paire et impaire).

Tableau 7

Décimal	BCD 8421	Parité Paire	Parité Impaire
0	0000	0000 0	0000 1
1	0001	0001 1	0001 0
2	0010	0010 1	0010 0
3	0011	0011 0	0011 1
4	0100	0100 1	0100 0
5	0101	0101 0	0101 1
6	0110	0110 0	0110 1
7	0111	0111 1	0111 0
8	1000	1000 1	1000 0
9	1001	1001 0	1001 1

Exemple: soit à déterminer les erreurs dans les valeurs suivantes:

Mots	Bit de parité	Type de parité
1001	0	impaire
11010	0	impaire
100101	0	paire
1010	0	paire
1001100	1	paire

Dans les exemples précédents, le premier et le troisième sont incorrects.

b. Détection et correction d'erreur

Le bit de parité permet de détecter l'erreur mais ne perme pas sa correction. Pour se faire une méthode très utilisé consiste à ajouter un mot de parité.

Exemple 1:

bit de parité

```
01111001
00000111
10001111
01101110
00110001
01011000
11110001
11111000   ← mot de parité
```

Exemple 2: Détermination et correction d'erreur

```
0111  1  001
0000  0  111
1000  1  111
0110  1  110
0011  1  001   ← erreur de parité détecté
0101  1  000
1111  0  001
1111  1  000
erreur de parité détecté ↑
```

c. Détection et correction d'erreur sur un seul mot binaire

Considérons un mot binaire à 4 bits

b3 b2 bl b0

La parité doit être déterminée pour les trois groupes de 3 bits:

Pl détermine la parité de **b3 b2 bl**
P2 détermine la parité de **b3 bl b0**
P3 détermine la parité de **b2 bl b0**

L'information à transmettre sera alors : **P3 P2 P1 b3 b2 bl b0**

A la réception il peut y avoir des erreurs dans n'importe quel bit parmi le mot à 7 bits.

Si par exemple la parité n'est pas vérifiée pour P2 et P3 alors l'erreur s'est produite dans b0.

Le tableau suivant résume les possibilités d'erreur pour différents cas de parité non vérifiée.

Parité non vérifiée	Erreur
P2 P3	b0
Pl P2 P3	bl
Pl P3	b2
Pl P2	b3
P3	P3
P2	P2
Pl	Pl

Exemple: Soit à détecter l'erreur dans un mot binaire: 1 1 1 0 1 0 0 (parié impaire)

Solution: 1 1 1 0 1 0 0
P3 P2 Pl b3 b2 bl b0

En utilisant les définitions des parités de P3 , P2 et Pl on trouve: Pl = 0, P2 = 1 et P3 = 0. La parité n'est pas vérifiée pour Pl et P3. D'après le tableau l'erreur se trouve dans b0.

C.6. Code de gray

Il est dit également code binaire réfléchi; il est surtout utilisé pour coder des positions. on rencontre ses applications dans le domaine optique et mécanique. Le code de Gray est un code non pondéré et il est caractérisé par le changement d'un seul bit entre deux mots successifs.

Le tableau 8 présente le code de Gray pour des valeurs comprises entre 0 et 12.

Tableau 8

décimal	C.B.Naturel	Code de Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010

TD - AII, Série n°1

EX.1

Convertir en base 2 les nombres décimaux suivants:
0,375 ; 17,150 3,14 ; 6,28

EX.2

Convertir en base 10 les nombres binaires suivants:
1011 ; 10110 ; 101,1
commenter les résultats.

EX.3

Déterminer le complément à 1 et à 2 des nombres binaires suivants:
10101,01 ; 0111000 ; 1,011 ; 10000, 00000

EX.4

Faire la soustraction en utilisant le complément à 2 et à 1.:

11010	11010	10010	111100
- 11101	- 10000	- 10011	- 110000
=	=	=	=

EX.5

Trouver le nombre de digit nécessaires pour représenter un nombre décimal compris entre 1-1000 en: binaire, octal, hexadécimal, BCD_{8421}.

EX.6

Convertir dans la base 8: $(182)_{10}$ $(1011110101)_2$ $(47,75)_{10}$

EX.7

Convertir dans la base 16: $(4732)_{10}$ $(1011100111000111)_2$ $(254,03125)_{10}$

EX.8

Convertir dans la base BCD_{8421}: $(4723)_{10}$ $(1010111011)_2$ $(AA4E)_{16}$

EX.9

Détecter et corriger l'erreur dans les mots suivants:

bit de parité (paire)	bit de parité
0001 1	0110 1
1011 0	1000 0
1100 0	1100 1
0000 0	1111 0
0010 1	0011 1
mot de parité (paire)	mot de parité (impaire)

EX.10

Convertir les nombres ci-après suivant la représentation module et signe sur un format de 8 bits (signe compris): 19 ; -52 ; -127 ; -128

EX.11

Avec n bits, quel est le nombre décimal le plus grand représentable suivant la représentation module et signe?

EX.12

Avec n bits, quel est le nombre décimal le plus petit représentable suivant la représentation module et signe?

EX.13

Soit le nombre N= $n_7\ n_6\ n_5\ n_4\ n_3\ n_2\ n_1\ n_0$ représenté en module et signe (n_7 est donc le bit de signe. On envoie ce nombre sur un convertisseur numérique analogique fonctionnant en binaire pur. La tension de sortie de ce convertisseur est donc:

Vs = Vref ($n_7\ 2^7 + n_6\ 2^6 + n_5\ 2^5 + n_4\ 2^4 + n_3\ 2^3 + n_2\ 2^2 + n_1\ 2^1 + n_0\ 2^0$)

Quel est alors la loi d'évolution de Vs en fonction de N sur l'intervalle [Nmin, Nmax]?

Que remarque-t-on ?

EX.14

a)Convertir les nombres ci-après suivant la représentation complément à 2 sur un format de 8 bits (signe compris): 19 ; -52 ; -127 ; -128

b)Avec n bits, quel est le nombre décimal le plus grand représentable suivant la représentation complément à 2?

c)Avec n bits, quel est le nombre décimal le plus petit représentable suivant la représentation complément à 2?

d)Soit le nombre N= $n_7\ n_6\ n_5\ n_4\ n_3\ n_2\ n_1\ n_0$ représenté en complément à 2 (n_7 est donc le bit de signe. On envoie ce nombre sur un convertisseur numérique analogique fonctionnant en binaire pur. La tension de sortie de ce convertisseur est donc:

Vs = Vref ($n_7\ 2^7 + n_6\ 2^6 + n_5\ 2^5 + n_4\ 2^4 + n_3\ 2^3 + n_2\ 2^2 + n_1\ 2^1 + n_0\ 2^0$)

-Quel est alors la loi d'évolution de Vs en fonction de N sur l'intervalle [Nmin, Nmax]?

-Que remarque-t-on ?

-Comment peut-on rendre la loi d'évolution de Vs linéaire et croissante?

EX.15

a)Convertir le nombre $(8620)_{10}$ en BCD_{8421} puis en Excédent 3.

EX.16

Tracer un tableau à 16 lignes avec le code binaire naturel (0-15 en décimal) et son équivalent avec le code binaire réfléchi (code de Gray).

A partir de tableau, établir une règle générale permettant la conversion du code binaire naturel au code de Gray et inversement. En déduire les conversions suivantes :

$$(1010111011)_2 = (\ ?)_{Gray} \quad ; \quad (1000111011)_{Gray} = (\ ?)_2$$

EX.17

Déterminer les tables avec les codes BCD de pondération 3321 et 443-2 de telle sorte que le complément à 9 de chaque nombre décimal soit obtenu par le complément logique du nombre en code BCD.

EX.18

Déterminer le complément à 9 et à 10 des nombres décimaux suivants :

13579 ; 9900 ; 80090 ; 10000

EX.19

Déterminer le complément à 1et à 2 des nombres binaires suivants :

1101 ; 0,101 ; 10,11 ; 100,001 , 1,000001

EX.20

Faire la soustraction en utilisant le complément à 2 et à 1:

11010,11	11101	1101,101	00000,0101
- 11101,01	- 11011	- 1111,010	- 11101,1101
=	=	=	=

Chapitre II

FONCTIONS LOGIQUES COMBINATOIRES

A. Minimisation des fonctions logiques

Minimiser une fonction revient à diminuer le nombre de terme qui intervient dans sa définition et ainsi on réduit le nombre de circuits nécessaires à sa réalisation.

A.1. Méthode algébrique de simplification

On utilise les règles et les postulats de l'algèbre de boole.

Exemple :

$$F_1 = x + \bar{x}.y = \left(x + \bar{x}\right).(x + y) = 1.(x + y) = x + y$$

$$F_2 = x.\left(\bar{x} + y\right) = x.\bar{x} + x.y = 0 + x.y = x.y$$

$$F_3 = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y} = \bar{x}.z\left(y + \bar{y}\right) + x.\bar{y} = \bar{x}.z + x.\bar{y}$$

$$F_4 = x.y + \bar{x}.z + y.z = x.y + \bar{x}.z + y.z\left(x + \bar{x}\right)$$

$$= x.y + \bar{x}.z + y.z = x.y + \bar{x}.z + y.z.x + y.z.\bar{x}$$

$$= x.y(1 + z) + \bar{x}.z(1 + y) = x.y + \bar{x}.z$$

$$F_5 = (x + y).\left(\bar{x} + y\right)(z + y) = (x + y).\left(\bar{x} + z\right)$$

A.2. Méthode graphique de Karnaugh

C'est une méthode systématique et pratique lorsque le nombre de variables est inférieur à 5.

Une fois que les valeurs de la fonction logique sont introduites dans le tableau, on cherche à simplifier la fonction en regroupant les « 1 » ou les « 0 » qui se trouvent dans des carrés adjacents dans les boucles les plus larges. On peut faire des groupements de 2^n carrés adjacents avec n = 0, 1, 2, 3.....etc.

Une variable est éliminée de l'expression logique si elle se présente dans la boucle sous forma directe et inversée (en effet la variable + le complément de la variable = 1).

Les variables qui sont les mêmes pour tous les carrés de la boucle doivent figurer dans l'expression finale.

Remarque La dernière ligne est adjacente à la première et la dernière colonne est adjacente à la première.

Exemple :

AB \ C	0	1
00	1	0
01	0	0
11	0	0
10	1	0

$$X(A,B,C) = \bar{B}.\bar{C}$$

AB \ C	0	1
00	0	1
01	0	1
11	0	1
10	0	1

$$X(A,B,C) = C$$

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$$X(A,B,C,D) = \bar{A}.\bar{B}.C + A.\bar{B}.\bar{D}$$

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$$X(A,B,C,D) = \bar{B}.\bar{D}$$

AB \ CD	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	1	1	1	0
10	0	0	1	0

$$X(A,B,C,D) = B.C.D + A.C.D + A.B.C + A.C.D$$

CD / AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

$$X(A,B,C,D) = B$$

CDE / AB	000	001	011	010	110	111	101	100
00	1	0	0	1	1	0	0	1
01	0	1	1	0	0	1	1	0
11	0	1	1	0	0	1	1	0
10	0	1	0	0	0	0	1	0

$$X(A,B,C,D,E) = B.E + A.\overline{D}.E + \overline{A}.\overline{B}.\overline{E}$$

Pour trouver l'expression en Nand (ON) d'une fonction logique on écrit son développement en somme de produits (dans le tableau de Karnaugh on fait des groupement de « 1 ») et l'on remplace les signes opératoires OU et ET par le signe opératoire Nand.

Pour trouver l'expression en NOR (NI) d'une fonction logique on fait son développement en produit de sommes (dans le tableau de Karnaugh on fait des groupements de « 0 ») et l'on remplace les signes opératoires OU et ET par le signe opératoire NOR.

Exemple : Soit $F(A,B,C,D) = \{0,\ 1,\ 2,\ 5,\ 8,\ 9,\ 10\}$

CD / AB	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

$$F(A,B,C,D) = \overline{B}.\overline{C} + \overline{B}.\overline{D} + \overline{A}.\overline{C}.D$$
$$= \left(\overline{B}/\overline{C}\right) / \left(\overline{B}/\overline{D}\right) / \left(\overline{A}/\overline{C}/D\right)$$

De même :

$$\overline{F}(A,B,C,D) = C.D + A.B + B.\overline{D}$$

$$F(A,B,C,D) = \overline{C.D + A.B + B.\overline{D}} = \left(\overline{C}\downarrow\overline{D}\right)\downarrow\left(\overline{A}\downarrow\overline{B}\right)\downarrow\left(\overline{B}\downarrow D\right)$$

il y a des applications où certaines combinaisons des variables d'entrée ne se produisent jamais. Le code BCD par exemple possède 6 combinaisons qui ne sont pas utilisées. un circuit digital utilisant ce code fonctionne avec l'hypothèse que ces combinaisons ne se produisent jamais à condition bien sûr que le système fonctionne correctement. comme conséquence de ceci la fonction de sortie est indifférente pour ces combinaisons de variables.

Par convention le symbole de l'indifférent est : X

Ces indifférents seront utilisés pour simplifier davantage la fonction logique comme le montre l'exemple suivant :

A	**B**	**C**	**Z**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	X
1	0	0	X
1	0	1	1
1	1	0	1
1	1	1	1

AB \ **C**	**0**	**1**
00	0	0
01	0	X
11	1	1
10	X	1

$Z = A$

indifférents

A.3. Méthode tabulaire de Quine-MC Cluskey

Le nombre de variables doit être supérieur à 5 ou à 6, pratique avec l'utilisation d'un ordinateur dont il y a deux parties :

- recherche des termes candidats à être inclus dans la fonction simplifiée : ces termes sont appelés : impliquant premiers.

- choix à travers les impliquant premiers de ceux qui donnent une expressions logique avec le moindre nombre de variables.

Exemple :

Soit $F(W,X,Y,Z) = \{1,\ 4,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 15\}$

D'abord détermination des impliquant premiers

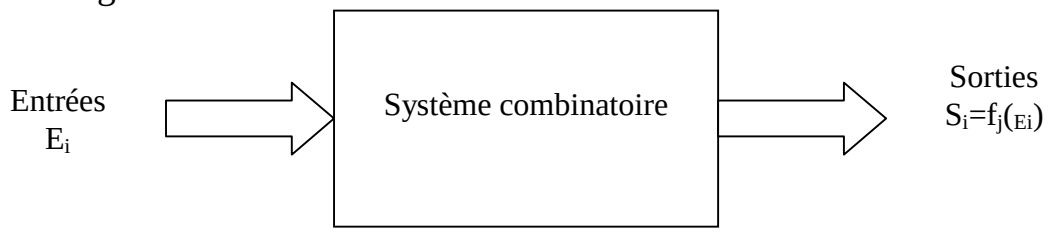
WXYZ (a)					(b) WXYZ		(c) WXYZ	
0	0	0	1	1	1,9	-001	8, 9, 10, 11	10--
0	1	0	0	4	4, 6	01-0		
1	0	0	0	8	8, 9	100-0		
0	1	1	0	6	8, 10	10-0		
1	0	0	1	9	6, 7	011-		
1	0	1	0	10	9, 11	10-1		
0	1	1	1	7	10, 11	101-		
1	0	1	1	11	7, 15	-111		
1	1	1	1	15	11, 15	1-11		

Les impliquant premiers sont les termes qui ne sont pas marqués par une trame de fond.

$$* \text{ -001 : } \bar{X}.\bar{Y}.Z$$
$$* \text{ 01-0 : } \bar{W}.X.\bar{Z}$$
$$* \text{ 011- : } \bar{W}.X.Y$$
$$* \text{ -111 : } X.Y.Z$$
$$* \text{ 1-11 : } W.Y.Z$$
$$* \text{ 10-- : } W.\bar{X}$$

B. Fonctions combinatoires usuelles et principaux types de circuits MSI

Les fonctions combinatoires fondamentales sont des problèmes combinatoires complexes communs à nombreux utilisateurs, réalisables dans des circuits intégrés.



B.1. Comparaison entre information

Il s'agit d'un circuit logique combinatoire qui compare deux grandeurs binaires et produit des sorties désignant l'égalité de ces grandeurs ou lequel est le plus grand ou le plus petit.

Les comparateurs de grandeurs sont employés assez intensivement dans les circuits de décodage des adresses des ordinateurs. Ce sont eux qui permettent de sélectionner le périphérique ou de localiser la zone mémoire contenant les données que l'on veut trouver. Ces éléments comparent le code d'adresse

envoyé par le processeur central à un code d'adresse matériel ; si les deux coïncident, la sortie $S_{A=B}$ du comparateur active le dispositif ayant l'adresse correspondante.

Les comparateurs sont également très utiles dans les applications de régulation où un nombre binaire représentant une variable physique (comme la vitesse, la position, le courant, etc…) est comparé à une valeur de consigne. Le sorties du comparateur servent à l'envoi de signaux pour la conduite des mécanismes qui ramènent la variable physique vers son point de consigne. De même, les sorties du comparateur peuvent servir comme déclencheurs d'alarmes en cas d'anomalies dans certaines applications.

B.1.1. Principe

Soient a_i et b_i deux éléments binaires et F1, F2 et F3 les trois sorties permettant de calculer respectivement ($a_i=b_i$) ou ($a_i>b_i$) ou ($a_i<b_i$):

a_i	b_i	F1 ($a_i=b_i$)	F2 ($a_i>b_i$)	F3 ($a_i<b_i$)
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

$F1=\overline{a_i \oplus b_i}=a_i \otimes b_i$: Fonction égalité

$F2=a_i\overline{b_i}$, $F3=\overline{a_i}b_i$

B.1.2. Extension pour n éléments binaires

Soient deux nombres binaires A et B représentés par n éléments binaires :

$$A=a_{n-1}a_{n-2}\cdots a_1a_0 \quad \text{et} \quad B=b_{n-1}b_{n-2}\cdots b_1b_0$$

$F1_{(A=B)}$=1 SI ($a_0=b_0$) ET ($a_1=b_1$) ET ….ET ($a_{n-1}=b_{n-1}$)

$$F1=\overline{(a_0 \oplus b_0)} \bullet \overline{(a_1 \oplus b_1)} \bullet \cdots \bullet \overline{(a_{n-1} \oplus b_{n-1})}$$

$F2_{(A>B)}$=1 SI ($a_{n-1}>b_{n-1}$) OU ($a_{n-1}=b_{n-1}$) ET ($a_{n-2}>b_{n-2}$) OU ….

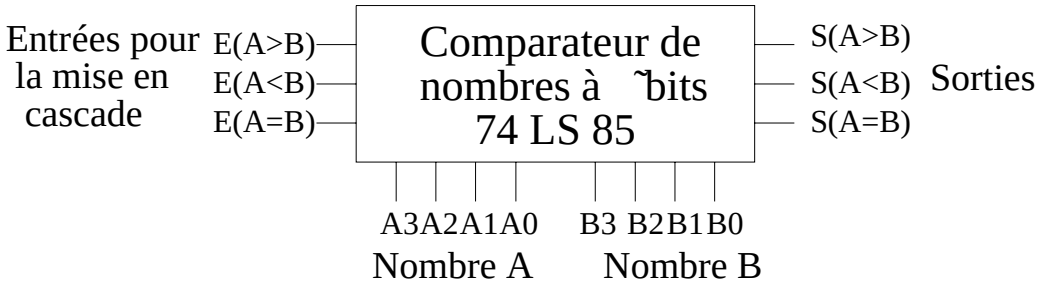
$$F2=a_{n-1}\overline{b_{n-1}}+\overline{(a_{n-1} \oplus b_{n-1})} \bullet a_{n-2}\overline{b_{n-2}}+\cdots$$

$F3_{(A<B)}$=1 SI ($a_{n-1}<b_{n-1}$) OU ($a_{n-1}=b_{n-1}$) ET ($a_{n-2}<b_{n-2}$) OU ….

$$F3=\overline{a_{n-1}}b_{n-1}+\overline{(a_{n-1} \oplus b_{n-1})} \bullet \overline{a_{n-2}}b_{n-2}+\cdots$$

B.1.3. Exemple de circuits MSI : 7485

C'est un comparateur de grandeurs à 4 éléments binaires. Les entrées de cascade permettent la comparaison de nombres de longueur quelconque par association des boîtiers entre eux. Le schéma synoptique ainsi que la table de fonctionnement de ce circuit sont données par les figures suivantes :



D'autres circuits MSI assurent la fonction de comparaison de grandeurs binaires telle que:

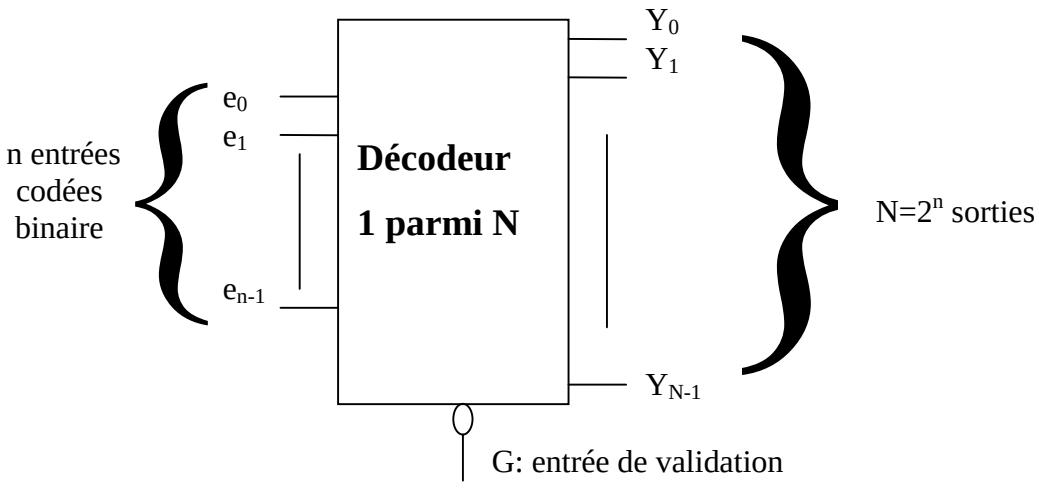
* 74 LS 682: Comparateur non cascadable de deux nombres à 8 eb.
* 74 LS 686: Comparateur cascadable de deux nombres à 8 eb.

B.2. décodage binaire

B.2.1. Principe

Une quantité codée en binaire avec n eb peut représenter $N=2^n$ valeurs. La fonction de décodage binaire consiste à transformer un codage binaire naturel (ou pur) en codage 1 parmi N (avec $N=2^n$).

Le décodeur binaire est un circuit à n entrées, $N=2^n$ sorties et éventuellement une ou plusieurs entrées G validant les fonctions de sortie.



Si G=1, toutes les sorties de décodeur sont inactives quelque soit l'état de l'information binaire sur $e_0\, e_1 \cdots e_{n-1}$.

Si G=0, pour une valeur binaire i sur les entrées $e_0 e_1 \cdots e_{n-1}$, la sortie de rang i est active et toutes les autres sont inactives. L'équation d'une sortie de rang i s'écrit: $Y_i = \overline{G}(e_0 e_1 \cdots e_{n-1} = i)$; $i : 0 \ldots 2^{n-1}$.

B.2.2. Exemple de circuits MSI : 74 138

Le circuit 74 138 est un décodeur 3 vers 8 qui répond aux principes généraux précédents. Les figures suivantes en donnent respectivement le schéma synoptique et la table de vérité.

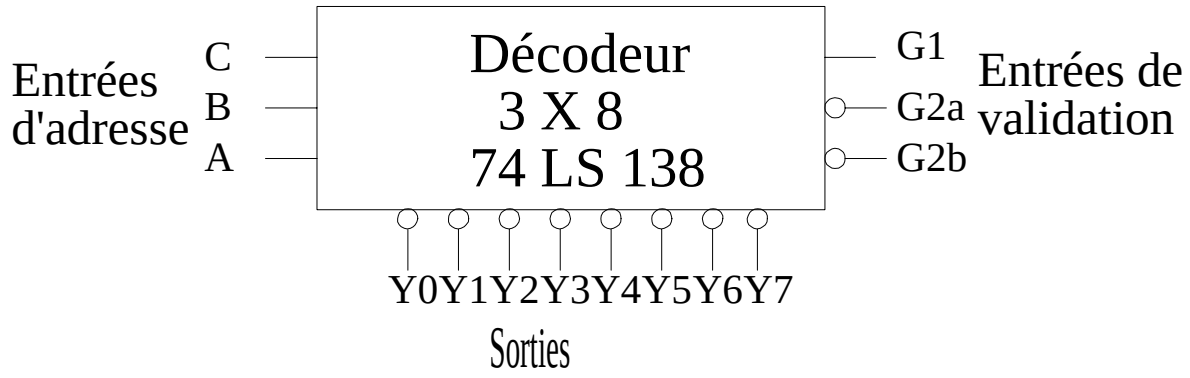


Table de vérité

Entrées		Sorties
G_1 G_{2a} G_{2b}	C B A	Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7
1 0 0	0 0 0	0 1 1 1 1 1 1 1
1 0 0	0 0 1	1 0 1 1 1 1 1 1
1 0 0	0 1 0	1 1 0 1 1 1 1 1
1 0 0	0 1 1	1 1 1 0 1 1 1 1
1 0 0	1 0 0	1 1 1 1 0 1 1 1
1 0 0	1 0 1	1 1 1 1 1 0 1 1
1 0 0	1 1 0	1 1 1 1 1 1 0 1
1 0 0	1 1 1	1 1 1 1 1 1 1 0
X 1 X	X X X	1 1 1 1 1 1 1 1
X X 1	X X X	1 1 1 1 1 1 1 1
0 X X	X X X	1 1 1 1 1 1 1 1

D'autres circuits MSI assurent la fonction de décodage binaire telle que:

* 74 LS 154: Décodeur 16 sorties avec deux entrées de validation (boîtier 24 broches)

* 74 LS 42: Décodeur 10 sorties (boîtier 16 broches).

* 74 LS 139: Double décodeur à 4 sorties avec entrées de validation (boîtier 16 broches, deux décodeurs indépendants).

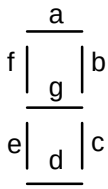
* 74 LS 155 (74 LS 156): Double décodeur à 4 sorties à adresse unique et double validation (boîtier 16 broches).

B.3. décodage BCD-7 segments

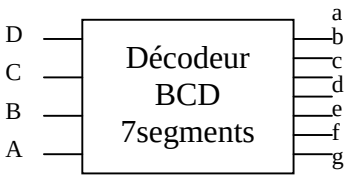
B.3.1. Principe

Dans de nombreux affichages numériques, les dix chiffres du code décimal (0 à 9), et parfois les caractères hexadécimaux (A à F), sont configurés au moyen de 7 segments. Chaque segment est constitué d'un matériau qui émet la lumière quand il est traversé par un courant. Les matériaux les plus utilisés sont les diodes électroluminescentes (LED) et les filaments incandescents.

La fonction de décodage consiste à traduire la série d'entrée de valeurs représentées par les quatre bits du code BCD(8421) en sept sorties correspondant aux valeurs des sept segments de l'afficheur. Les sorties actives permettent de faire passer un courant dans le segments qui forment le chiffre décimal correspondant au quatre bits du BCD.



4 entrées d'un nombre codé



Décimal	BCD (8421)				Sorties						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

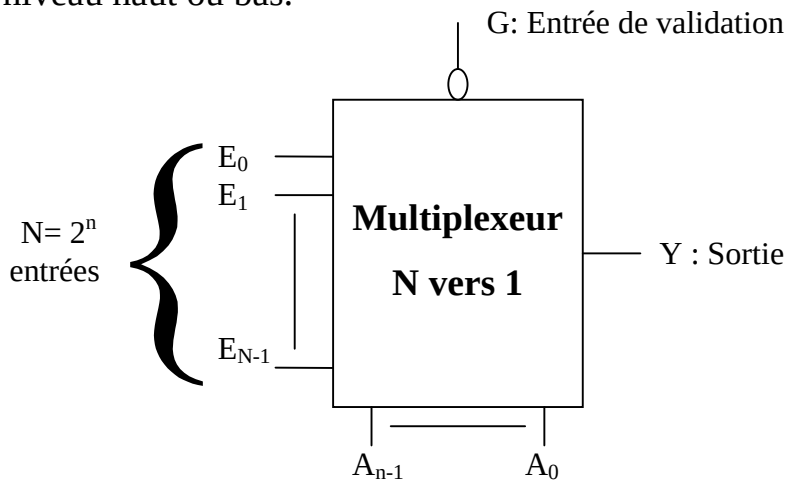
B.3.2. Exemple de circuits MSI :

Comme exemple de décodeur de BCD à sept segment, citons les circuits intégrés 7442, 7445, 7446, 7447, 7448, 7449 et 74247. Parmi les caractéristiques techniques de ces décodeurs, relevons l'affichage de signes particuliers en plus des nombres décimaux.

B.4. Multiplexage de données

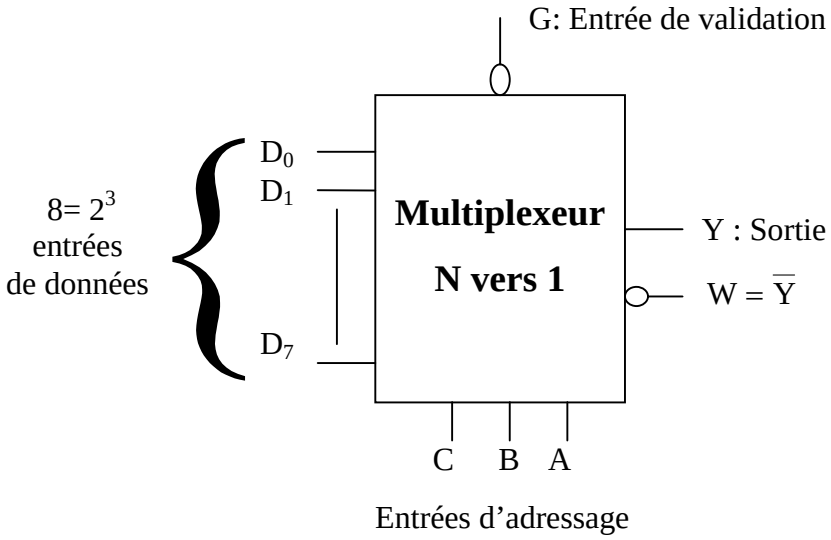
B.4.1. Principe

Le multiplexeur est un circuit qui permet de sélectionner une ligne d'entrée par une adresse et de faire apparaître à la sortie l'état de cette ligne, c'est à dire le niveau haut ou bas.



Un multiplexeur se comporte comme un commutateur dans lequel un code numérique appliqué aux entrées d'adressage commande les entrées de données qui sont raccordées à la sortie. Autrement dit, un multiplexeur choisit une source de données d'entrée parmi N et transmet celle-ci à la seule voie de sortie existante.

B.4.2. Exemple de circuits MSI :74151



Entrées		**|Sortie**
Addresse	Validation	
C B A	G	Y
X X X	1	0
0 0 0	0	D0
0 0 1	0	D1
0 1 0	0	D2
0 1 1	0	D3
1 0 0	0	D4
1 0 1	0	D5
1 1 0	0	D6
1 1 1	0	D7

B.5. l'Unité Arithmétique et Logique

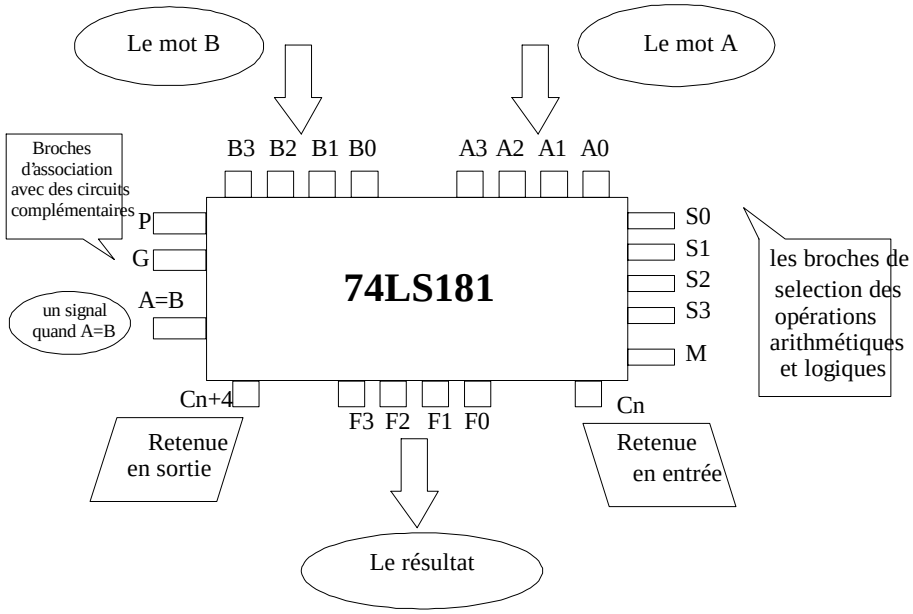
B.5.1. Principe

Nous nous rapprochons très sérieusement du microprocesseur avec ce composant. En effet, l'unité arithmétique et logique est un circuit qui permet de faire un certain nombre d'opérations entre deux groupes d'entrées et qui délivre un résultat sur un groupe de sortie.

B.5.2. Exemple de circuits MSI :74181

Supposons que notre UAL soit le circuit 74181: C'est une UAL à 4 bits, ce qui signifie qu'elle est capable d'effectuer des opérations sur des valeurs binaires jusqu'à 4 bits. Chaque groupe est un mot. De même, les UAL peuvent être branchées en cascade, avec 8, 16, 32 bits ...

Vous trouvez ci-joint le schéma synoptique, ainsi que le résumé de la table de vérité de l'UAL.



La table de fonctionnement est la suivante:

Sélection					***Fonction***
S4	**S3**	**S2**	**S1**	**S0**	**F**
0	0	0	0	0	A
0	0	0	0	1	A+B
0	0	0	1	0	$A + \overline{B}$
0	0	0	1	1	0 moins 1
0	0	1	0	0	A plus $A\overline{B}$
0	0	1	0	1	(A+B) plus $A\overline{B}$
0	0	1	1	0	A moins B moins 1
0	0	1	1	1	$A\overline{B}$ moins 1
0	1	0	0	0	A plus AB
0	1	0	0	1	A plus B
0	1	0	1	0	$(A + \overline{B})$ plus AB
0	1	0	1	1	AB moins 1
0	1	1	0	0	A plus A
0	1	1	0	1	(A+B) plus A
0	1	1	1	0	$(A + \overline{B})$ plus A
0	1	1	1	1	A moins1
1	0	0	0	0	$\overline{A}$
1	0	0	0	1	$\overline{A + B}$
1	0	0	1	0	$\overline{A}B$
1	0	0	1	1	0
1	0	1	0	0	$\overline{AB}$
1	0	1	0	1	$\overline{B}$
1	0	1	1	0	$A \oplus B$
1	0	1	1	1	$A\overline{B}$
1	1	0	0	0	$\overline{A} + B$
1	1	0	0	1	$\overline{A \oplus B}$
1	1	0	1	0	B
1	1	0	1	1	AB
1	1	1	0	0	1
1	1	1	0	1	$A + \overline{B}$
1	1	1	1	0	A+B
1	1	1	1	1	A

« + » pour le OU logique, « plus » pour l'addition et « moins » pour la soustraction

B.5.3. Réalisation d'un additionneur et d'un soustracteur:

L'une des fonctions importantes de l'UAL est l'addition, pour savoir le noyau qui a fourni à la fin ce CI très important, on va essayer de déterminer un circuit simulant cette fonction arithmétique.

(a) Donner, en utilisant des portes logiques simples, le logigramme du demi additionneur sachant que ce dernier calcule la somme S de deux bits a et b et leur retenu C sans tenir compte du retenu de l'étage précèdent.

(b) Pour obtenir un additionneur complet de deux bits a_i et b_i qui calcule la somme S_i et le retenu C_i, il suffit de tenir compte du retenu de l'étage précèdent C_{i-1} comme entrée supplémentaire. Donner le logigramme correspondant.

(c) La somme de deux nombres quelconques étant basées sur les circuits précédents mis en cascade. Donner le logigramme réalisant la somme de deux nombres de deux bits chacun.

(d) Refaites les mêmes étapes pour réaliser la soustraction de deux nombres de deux bits chacun.

(e) En analysant les deux circuits (additionneur de deux nombres et soustracteur de deux nombres), on remarque que la différence est très faible (inverser ou non quelques variables), prévoir une entrée supplémentaire qui sélectionne le type de la fonction désirée (addition ou soustraction de deux nombres de deux bits chacun) en utilisant le minimum de portes logiques. Donner un logigramme complet avec une table de vérité décrivant le fonctionnement.

TD - AII, Série n°2

EX.1

Montrer l'égalité en utilisant l'algèbre de Boole

$\overline{(A+B)(\overline{\overline{A}\cdot\overline{B}})}=\overline{A}\cdot\overline{B}$; $(A+\overline{B})\cdot(\overline{A}+\overline{B}+C)=AC+\overline{B}$; $\overline{(A\oplus B)}+(B\oplus C)+(A\oplus C)=1$

EX.2

Montrer que les opérateurs NOR et NAND sont des opérateurs universels (c.à.d. on peut réaliser les trois fonctions logiques de base OU , ET et NON uniquement avec ces opérateurs).

EX.3

Soient les fonctions logiques suivantes :

$F_1(x,y,z,t) = \Sigma\ (3,4,5,7,9,13,14,15)$; $F_2(x,y,z,t) = \Pi\ (2,3,4,5,6,7)$

$F_3(x,y,z,t) = \Sigma\ (3,4,5,6,7,8,9,10,12,13)$; $F_4(x,y,z,t) = \Sigma\ (0,1,2,3,4,5,8,9,10,11)$

$F_5(x,y,z) = \Sigma\ (0,2,3,4,6)$; $F_6(x,y,z,t) = \Pi\ (0,1,2,8,9,12,13,14,15)$

$F_7(x,y,z,t) = \Sigma\ (3,4,6,8,10,14)$; $F_8(x,y,z,t) = \Pi\ (1,3,7,12,15)$

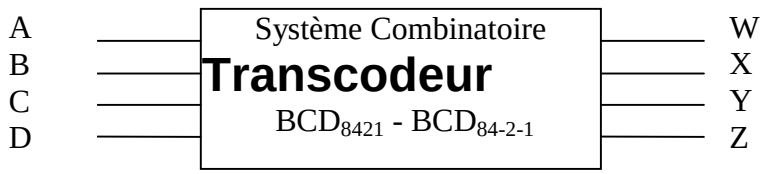
a)Déterminer les équations simplifiées de F1, F2 et F7 sous forme de somme de Produit (1ère forme canonique simplifiée) et de produit de somme (2ème forme canonique simplifiée)

b)Trouver la forme minimale avec des portes NAND des fonctions F3 et F4

c)Trouver la forme minimale avec des portes NOR des fonctions F5 et F6

EX.4

Déterminer le logigramme d'un transcodeur du BCD_{8421} de variables (ABCD) en $BCD_{84\text{-}2\text{-}1}$ de variables (W,X,Y,Z) avec le minimum de portes NAND à deux entrées.



EX.5

Réaliser un système modulaire permettant avec 8 modules identiques d'assurer la comparaison de deux mots binaires A et B de 8 bits chacun. Le système doit indiquer si A>B, A=B ou A<B..

EX.6

Donner les schémas de câblage des fonctions logiques suivantes avec un décodeur 1 parmi 8 et des portes NAND.

$F_1(x, y, z) = xy + \bar{x}z$; $F_2(x,y,z,t) = \Sigma\ (0,1,4)$; $F_3(x,y,z,t) = \Sigma\ (3,4,5,7)$

EX.7

Donner les schémas de câblage des fonctions logiques suivantes avec des multiplexeurs 8 vers 1

$$F_1(x,y,z,t) = \Sigma\ (3,4,5,7,9,13,14,15)$$

$$F_2(x,y,z,t) = \Pi\ (2,3,4,5,6,7)$$

$$F_3(x,y,z,t) = \Sigma\ (3,4,5,6,7,8,9,10,12,13)$$

$$F_4(x,y,z,t) = \Sigma\ (0,1,2,3,4,5,8,9,10,11)$$

EX.8 : Réalisation d'un additionneur et d'un soustracteur:

L'une des fonctions importantes de l'UAL est l'addition, pour savoir le noyau qui a fourni à la fin ce CI très important, on va essayer de déterminer un circuit simulant cette fonction arithmétique.

(a) Donner, en utilisant des portes logiques simples, le logigramme du demi additionneur sachant que ce dernier calcule la somme S de deux bits a et b et leur retenu C sans tenir compte du retenu de l'étage précèdent.

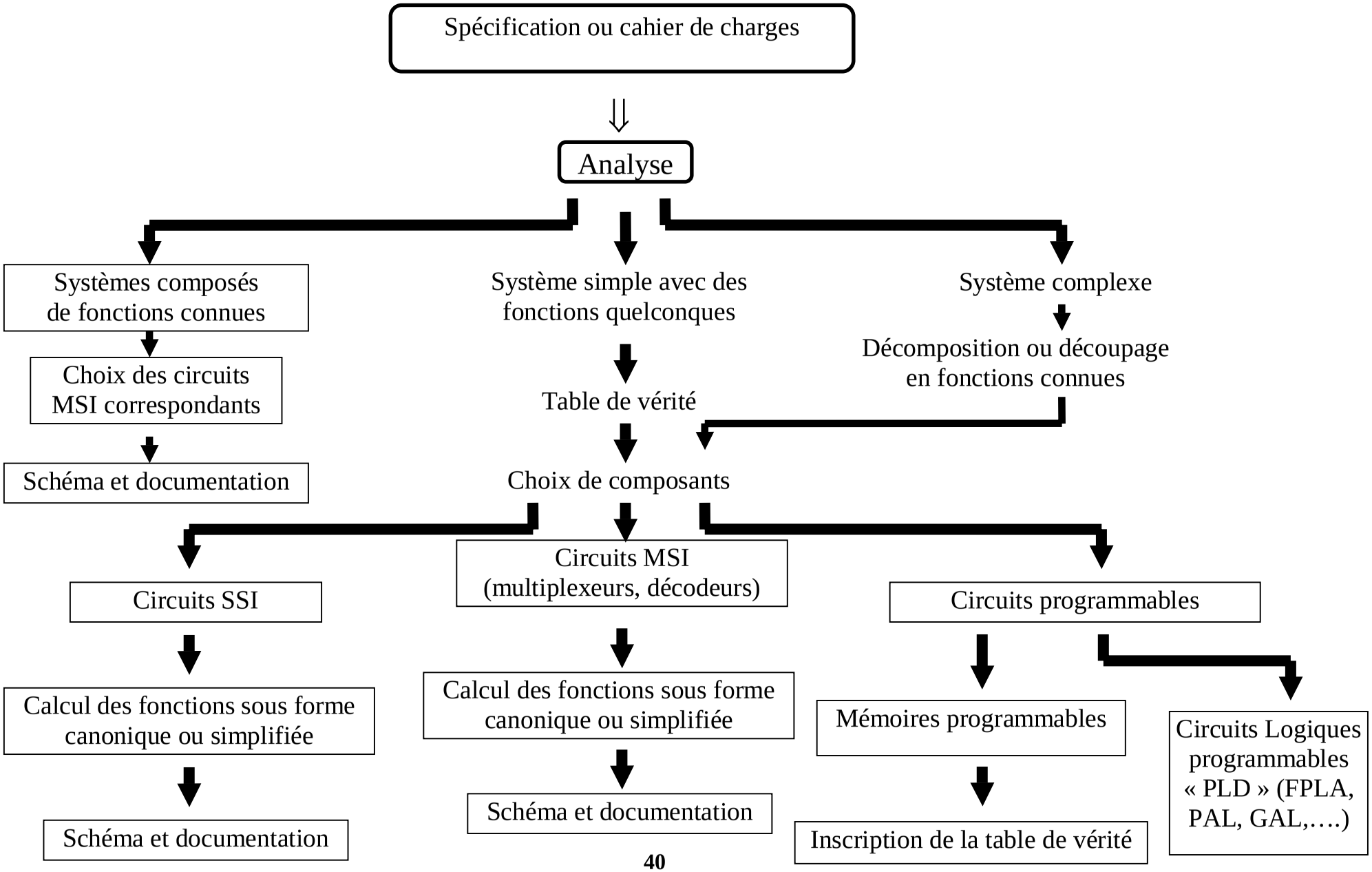
(b) Pour obtenir un additionneur complet de deux bits a_i et b_i qui calcule la somme S_i et le retenu C_i, il suffit de tenir compte du retenu de l'étage précèdent C_{i-1} comme entrée supplémentaire. Donner le logigramme correspondant.

(c) La somme de deux nombres quelconques étant basées sur les circuits précédents mis en cascade. Donner le logigramme réalisant la somme de deux nombres de deux bits chacun.

(d) Refaites les mêmes étapes pour réaliser la soustraction de deux nombres de deux bits chacun.

(e) En analysant les deux circuits (additionneur de deux nombres et soustracteur de deux nombres), on remarque que la différence est très faible (inverser ou non quelques variables), prévoir une entrée supplémentaire qui sélectionne le type de la fonction désirée (addition ou soustraction de deux nombres de deux bits chacun) en utilisant le minimum de portes logiques. Donner un logigramme complet avec une table de vérité décrivant le fonctionnement.

Approches de synthèse d'un système combinatoire



ISEFC BARDO A.U :2003/2004

Session de Septembre 2004

Epreuve : GE 206 – GM 206
Durée : 2*h* documents non autorisés

Exercice n°1 :

Soient les deux fonctions logiques suivantes :

$$F(a,b,c,d,e) = \sum(0,1,2,3,8,9,10,11)_1 + \Pi(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,20,21,22,23,24,26,28,29,30)_0$$

1) Dresser le tableau de Karnaugh correspondant à la fonction *F* et y inscrire les groupements permettant de déterminer l'expression la plus simple de *F*.

2) Ecrire alors l'expression minimale sous forme de somme de mintermes de *F* avec des portes logiques de base.

3) Donner le schéma à contacts correspondant.

4) Donner le logigramme correspondant.

5) Ecrire alors l'expression minimale de *F* avec des opérateurs NAND à 2 entrées uniquement.

Exercice n°2 :

La figure 2 représente un circuit combinatoire qui calcul la différence (*D* = *A*-*B*) de 2 nombre binaires de 2 bits chacun. Il accepte en entrée les 2 nombres binaires de 2 bits $A = (a_1a_0)$ et $B = (b_1b_0)$ et fournit en sortie le nombre binaire $D = (d_2d_1d_0)$ dont le resultat est représenté par les bits d_1d_0 et d_2 représente le bit de signe ($d_2 = 1$ si le nombre est positif, $d_2 = 0$ si le nombre est négatif).

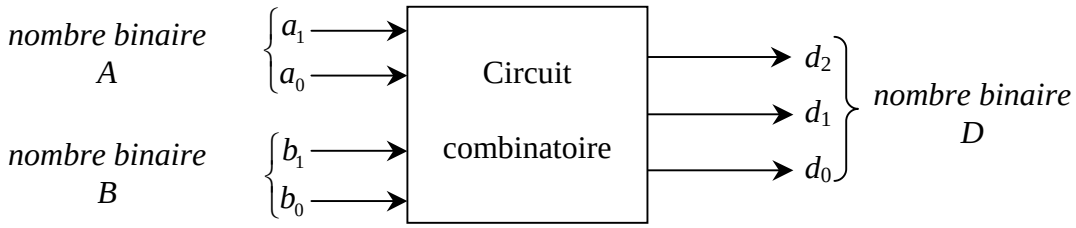


Fig 2

N.B : x_0 , y_0 et z_0 sont les bits de poids le plus faible.

1) Ecrire l'équation simplifiée de d_2 avec des portes logiques de base.

2) Etablir le schéma à contact correspondant.

3) Ecrire l'équation simplifiée de d_1 avec des portes logiques de base.

4) Etablir le logigramme correspondant.

5) Ecrire l'équation simplifiée de d_0 avec:

a. des opérateurs OU-excluif uniquement,

b. des opérateurs NAND à deux entrées uniquement,

c. des opérateurs NOR à deux entrées uniquement.

6) Tracer le logigramme de d_0.

Exercice n°3 :

Soit les nombres suivants : $m = (EF, F8)_{16}$ et $p = (231{,}13)_4$.

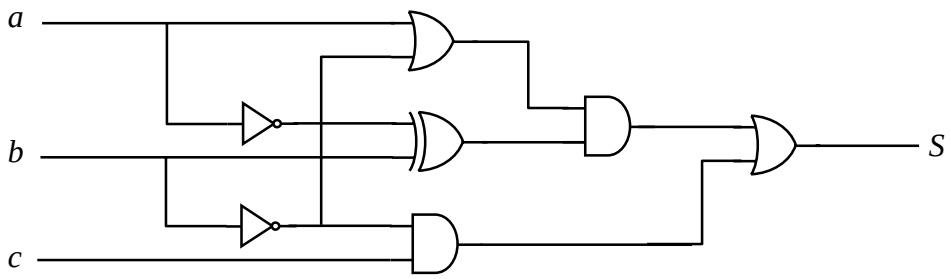
1) Convertir en binaire et en décimal les nombres *m* et *p*.

2) Réaliser la soustraction suivante en *binaire* et en utilisant la méthode du *complément à 2* (expliquer brièvement la méthode en précisant ses étapes) :

$$s = (231)_4 - (EF)_{16}$$

Exercice n°4 :

Soit le logigramme suivant :



1) Donner l'équation simplifiée avec des portes logiques de base de la sortie en fonction des entrées.

2) Donner le logigramme correspondant à cette équation avec des portes logiques NAND à 2 entrées seulement.

Bon travail

Session Mars 2004

Epreuve : GE 206 – GM 210
Durée : 2*h* documents non autorisés

Exercice n°1 :

La figure 2 représente un circuit qui calcul la fonction ($z = 2x + 3y$). Il accepte 2 nombres binaires de 2 bits x_1x_0 et y_1y_0 et fournit en sortie le nombre binaire $z_3z_2z_1z_0$ calculé à partir de la fonction arithmétique précédente.

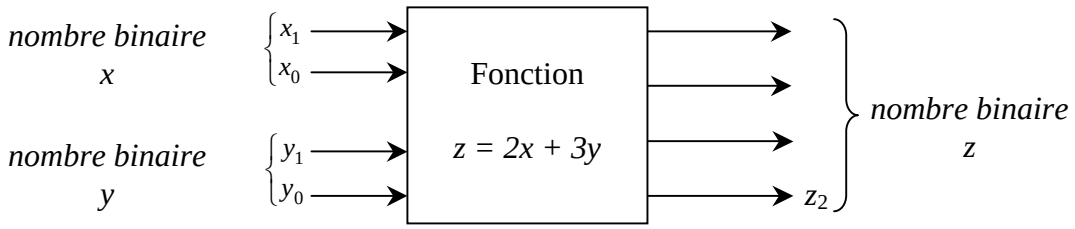


Fig 2

N.B : x_0 , y_0 et z_0 sont les bits de poids le plus faible.

7) Etablir la table de vérité traduisant le fonctionnement du circuit.

8) Ecrire les équations simplifiées de z_0 et z_1 avec des portes NAND à deux entrées seulement.

9) Ecrire l'équation de z_2 et z_3 avec des portes NOR à deux entrées seulement.

10) Tracer les logigrammes de z_0 ,z_1 , z_2 et z_3.

Exercice n°2 :

Soient les deux fonctions logiques suivantes :

$F(x,y,z,t) = \sum(0,1,2,5,7,8,10,13,15)_1$ $\qquad$ $G(x,y,z,t) = \prod(0,1,4,6,9,12,14)$

6) Ecrire les expressions simplifiés de ses deux fonctions sous forme de somme de produits (première forme canonique simplifier), puis sous forme de produit de sommes (deuxième forme canonique simplifier).

7) Ecrire l'expression minimale de *F* avec des opérateurs NAND à 2 entrées uniquement.

8) Ecrire l'expression minimale de *G* avec des opérateurs NOR à 2 entrées uniquement.

Exercice n°3 :

Soit les nombres suivants : $A = (79)_{10}$, $B = (94)_{10}$ et $C = (53)_{10}$.

1) Donner les codes de ses nombres en DCB(5421) (binaire naturel) et en DCB(44-2-1). (répondre sous forme de tableau).

2) Convertir le nombre $N_1 = \frac{A}{4}$ en binaire puis en octal (préciser le calcul).

3) Convertir le nombre $N_2 = \frac{B}{7}$ en hexadécimal (préciser le calcul).

N.B : La précision du calcul dans les question ***2)*** et ***3)*** est de trois chiffres après la virgule.

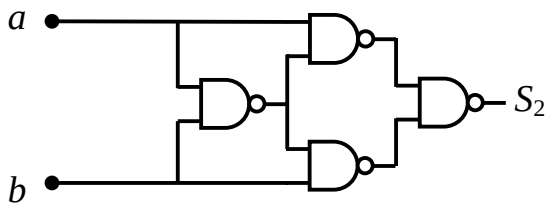
4) Réaliser les soustractions suivantes en *binaire* et en utilisant la méthode du *complément à 2* (expliquer la méthode en précisant toute ses étapes) :

$A - B = ?$; $B - C = ?$

5) Effectuer l'addition suivante dans la base 8 $(26)_8+(57)_8= (?)_8$

Exercice n°4 :

Soit le logigramme suivant :



3) Donner le logigramme équivalent en remplaçant chaque porte NAND par son équivalent en portes NOR.

4) Simplifier le logigramme obtenu (donner le résultat avec des portes NOR avec la normalisation française).

Bon travail

IESFC Bardo

Epreuve AU1 : GE/GM 206

Durée 2 H
Documents non autorisés

Exercice 1 :

Convertir les nombres suivants :

a- $(1221)_3 = (\ldots\ldots\ldots\ldots)_5$

b- $(111011)_2 = (\ldots\ldots\ldots\ldots..)_8$

c- $(1010/1110)_{BCD(8,4,2,1)} = (\ldots\ldots\ldots\ldots\ldots)_3$

d- $(1AC)_{16} = (\ldots\ldots\ldots\ldots\ldots..)_{BCD(8,4,2,1)}$

e- Réaliser la soustraction suivante par la méthode du complément à 2 $(25)_{10}$- $(53)_{10}$

Exercice 2 :

On se propose d'étudier un système de sécurité sur machine automatique.

Fonctionnement :
En marche normale, la mise en fonctionnement d'une machine automatique (M) impose l'ensemble des conditions suivantes :

- Contrôle du bon positionnement de la pièce par le capteur (p).
- Fermeture de l'écran de protection par un capteur (e).
- Action sur l'interrupteur (s).

En marche de réglage, cette machine fonctionne :

- Avec ou sans écran de protection.
- La pièce bien positionnée.
- Une clé engagée dans un contact à verrouillage (k).
- En actionnant sur l'interrupteur (s).

1)

a- Analyser le fonctionnement du système, en dressant la table de vérité.
b- Ecrire l'équation de la sortie (M) sous sa forme canonique complète.

2) Simplifier cette équation :

a- Algébriquement.
b- Graphiquement (tableau de Karnaugh).

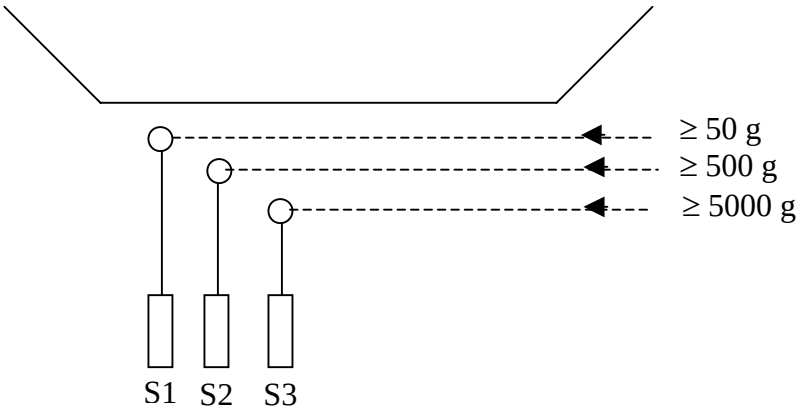
3)
a- Tracer le logigramme de M en utilisant les opérateurs logiques de base à deux entrées.
b- Etablir le schéma à contact.

4)
a- Transformer l'équation de M en utilisant des opérateurs NAND à deux entrées.
b- Tracer le logigramme correspondant.

Exercice 3 :

Un dispositif de pesage de paquets est constitué d'un plateau agissant sur trois mini capteurs s1, s2, et s3

- s1 change d'état binaire pour des paquets dont la masse est supérieure ou égale à 50 g.
- s2 change d'état binaire pour des paquets dont la masse est supérieure ou égale à 500 g.
- s3 change d'état binaire pour des paquets dont la masse est supérieure ou égale à 5000 g.



On dispose d'un pupitre indicateur par affichage lumineux.

H1	⊗	Masse < 50g
H2	⊗	Masse comprise entre 50 g et 500g
H3	⊗	Masse comprise entre 500 g et 5000g
H4	⊗	Masse ≥ 5000g

1- Dresser la table de vérité des sorties H1, H2, H3 et H4.

2- Simplifier les équations des sorties H1, H2, H3 et H4, algébriquement puis graphiquement.
3- Donner les logigrammes des sorties en utilisant que des opérateurs NOR à deux entrées.